文档主题

Nova 如何统计 OpenStack 计算资源？

为什么 free\_ram\_mb, free\_disk\_gb 有时会是负数？

即使 free\_ram\_mb, free\_disk\_gb 为负，为什么虚拟机依旧能创建成功？

资源不足会导致虚拟机创建失败，但指定了 host 有时却能创建成功？

**Nova 需统计哪些资源**

云计算的本质在于将硬件资源软件化，以达到快速按需交付的效果，最基本的计算、存储和网络基础元素并没有因此改变。就计算而言，CPU、RAM 和 DISK等依旧是必不可少的核心资源。

从源码和数据库相关表可以得出，Nova 统计计算节点的四类计算资源：

CPU: 包括 vcpus(节点物理 cpu 总线程数), vcpus\_used(该节点虚拟机的 vcpu 总和)

RAM: 包括 memory\_mb(该节点总 ram)，memory\_mb\_used(该节点虚拟机的 ram 总和)，free\_ram\_mb(可用 ram)   
Note: memory\_mb = memory\_mb\_used + free\_ram\_mb

DISK：local\_gb(该节点虚拟机的总可用 disk)，local\_gb\_used（该节点虚拟机 disk 总和），free\_disk\_gb(可用 disk)   
Note：local\_gb = local\_gb\_used + free\_disk\_gb

其它：PCI 设备、CPU 拓扑、NUMA 拓扑和 Hypervisor 等信息

**Nova 如何收集资源**

从 [源码](https://github.com/openstack/nova/blob/master/nova/virt/libvirt/driver.py#L4878) 可以看出，Nova 每分钟统计一次资源，方式如下：

CPU

vcpus: libvirt 中 get\_Info()

vcpu\_used: 通过 libvirt 中 dom.vcpus() 从而统计该节点上所有虚拟机 vcpu 总和

RAM

memory: libvirt 中 get\_Info()

memory\_mb\_used：先通过 /proc/meminfo 统计可用内存， 再用总内存减去可用内存得出(资源再统计时会重新计算该值)

DISK

local\_gb: os.statvfs(CONF.instances\_path)

local\_gb\_used: os.statvfs(CONF.instances\_path)(资源再统计时会重新计算该值)

其它

hypervisor 相关信息：均通过 libvirt 获取

PCI: libvirt 中 listDevices(‘pci’, 0)

NUMA: livirt 中 getCapabilities()

Nova-compute 在上报资源至数据库前，还根据该节点上的虚拟机又做了一次资源统计

**Nova 资源再统计**

首先分析为什么需要再次统计资源以及统计哪些资源。从 [源码](https://github.com/openstack/nova/blob/master/nova/compute/resource_tracker.py#L365) 可以发现，Nova 根据该节点上的虚拟机再次统计了 RAM、DISK 和 PCI 资源。

为什么需再次统计 RAM 资源？以启动一个 4G 内存的虚拟机为例，虚拟机启动前后，对比宿主机上可用内存，发现宿主机上的 free memory 虽有所减少(本次测试减少 600 MB)，却没有减少到 4G，如果虚拟机运行很吃内存的应用，可发现宿主机上的可用内存迅速减少 3G多。试想，以 64G 的服务器为例，假设每个 4G 内存的虚拟机启动后，宿主机仅减少 1G 内存，服务器可以成功创建 64 个虚拟机，但是当这些虚拟机在跑大量业务时，服务器的内存迅速不足，轻着影响虚拟机效率，重者导致虚拟机 shutdown等。除此以外，宿主机上的内存并不是完全分给虚拟机，系统和其它应用程序也需要内存资源。因此必须重新统计 RAM 资源，统计的方式为：   
free\_memory = total\_memory - CONF.reserved\_host\_memory\_mb - 虚拟机理论内存总和   
CONF.reserved\_host\_memory\_mb：内存预留，比如预留给系统或其它应用   
虚拟机理论内存总和：即所有虚拟机 flavor 中的内存总和

为什么要重新统计 DISK 资源？原因与 RAM 大致相同。为了节省空间， qemu-kvm 常用 [QCOW2](https://people.gnome.org/~markmc/qcow-image-format.html) 格式镜像，以创建 DISK 大小为 100G 的虚拟机为例，虚拟机创建后，其镜像文件往往只有几百 KB，当有大量数据写入时磁盘时，宿主机上对应的虚拟机镜像文件会迅速增大。而 os.statvfs 统计的是虚拟机磁盘当前使用量，并不能反映潜在使用量。因此必须重新统计 DISK 资源，统计的方式为：   
free\_disk\_gb = local\_gb - CONF.reserved\_host\_disk\_mb / 1024 - 虚拟机理论磁盘总和   
CONF.reserved\_host\_disk\_mb：磁盘预留   
虚拟机理论磁盘总和：即所有虚拟机 flavor 中得磁盘总和

当允许资源超配(见下节)时，采用上述统计方式就有可能出现 free\_ram\_mb, free\_disk\_gb 为负。

**资源超配与调度**

即使 free\_ram\_mb 或 free\_disk\_gb 为负，虚拟机依旧有可能创建成功。事实上，当 nova-scheduler 在调度过程中，某些 filter 允许资源超配，比如 CPU、RAM 和 DISK 等 filter，它们默认的超配比为：

CPU: CONF.cpu\_allocation\_ratio = 16

RAM: CONF.ram\_allocation\_ratio = 1.5

DISK: CONF.disk\_allocation\_ratio = 1.0

以 ram\_filter 为例，在根据 RAM 过滤宿主机时，过滤的原则为：   
memory\_limit = total\_memory \* ram\_allocation\_ratio   
used\_memory = total\_memory - free\_memory   
memory\_limit - used\_memory < flavor[‘ram’]，表示内存不足，过滤该宿主机；否则保留该宿主机

def host\_passes(self, host\_state, instance\_type):

"""Only return hosts with sufficient available RAM."""

requested\_ram = instance\_type['memory\_mb']

free\_ram\_mb = host\_state.free\_ram\_mb

total\_usable\_ram\_mb = host\_state.total\_usable\_ram\_mb

memory\_mb\_limit = total\_usable\_ram\_mb \* CONF.ram\_allocation\_ratio

used\_ram\_mb = total\_usable\_ram\_mb - free\_ram\_mb

usable\_ram = memory\_mb\_limit - used\_ram\_mb

if not usable\_ram >= requested\_ram:

LOG.debug("host does not have requested\_ram")

return False

宿主机 RAM 和 DISK 的使用率往往要小于虚拟机理论使用的 RAM 和 DISK，在剩余资源充足的条件下，libvirt 将成功创建虚拟机。

随想：内存和磁盘超配虽然能提供更多数量的虚拟机，当该宿主机上大量虚拟机的负载都很高时，轻着影响虚拟机性能，重则引起 qemu-kvm 相关进程被杀，即虚拟机被关机。因此对于线上稳定性要求高的业务，建议不要超配 RAM 和 DISK，但可适当超配 CPU。建议这几个参数设置为：

CPU: CONF.cpu\_allocation\_ratio = 4

RAM: CONF.ram\_allocation\_ratio = 1.0

DISK: CONF.disk\_allocation\_ratio = 1.0

RAM-Reserve: CONF.reserved\_host\_memory\_mb = 2048

DISK-Reserve: CONF.reserved\_host\_disk\_mb = 20480

**指定 host 创建虚拟机**

本节用于回答问题四，当所有宿主机的资源使用过多，即超出限定的超配值时(total\_resource \* allocation\_ratio)，nova-scheduler 将过滤这些宿主机，若未找到符合要求的宿主机，虚拟机创建失败。

创建虚拟机的 API 支持指定 host 创建虚拟机，指定 host 时，nova-scheduler 采取特别的处理方式：不再判断该 host 上的资源是否满足需求，而是直接将请求发给该 host 上的 nova-compute。

def get\_filtered\_hosts(self, hosts, filter\_properties,

filter\_class\_names=None, index=0):

"""Filter hosts and return only ones passing all filters."""

...

if ignore\_hosts or force\_hosts or force\_nodes:

...

if force\_hosts or force\_nodes:

# NOTE(deva): Skip filters when forcing host or node

if name\_to\_cls\_map:

return name\_to\_cls\_map.values()

return self.filter\_handler.get\_filtered\_objects()

当该 host 上实际可用资源时满足要求时，libvirt 依旧能成功创建虚拟机。

**最后，以一图总结本文内容**

